Security Protocols

CS 6750 Lecture 8

November 5, 2009

Riccardo Pucella

(Adapted from an old slide deck by Gavin Lowe)

Security Protocols

A security protocol is an exchange of messages between two or more agents, with security-relevant goals such as:

- establishing a secret cryptographic key
- achieving authentication
- guaranteeing anonymity

Modern everyday uses:

- financial transactions
- voting

Security Protocols

Designed to work in hostile environments, where the network is under the control of an hostile opponent who can:

- overhear messages
- intercept messages
- fake messages

Protocols normally use cryptography to achieve their security goals

- Opponent can encrypt or decrypt with keys he knows
- Maybe can try to cryptanalyze to break keys

We are often interested in protocol flaws that are independent of crypto: we therefore assume the cryptography is perfect

Cryptography: Notation

A message m can be encrypted with a cryptographic key k

$$\{m\}_k$$

If k is a symmetric key (a key for a shared key cryptosystem) known only to Alice and Bob, then they can exchange secret messages by encrypting them with k; this will also provide authentication and integrity

Public Key Cryptography

If PK(Alice) is Alice's public key, then Bob can send Alice a secret message by encrypting it with Alice's public key:

$$\{m\}_{PK(Alice)}$$

Alice can decrypt this message with her secret key SK(Alice)

Signature are often represented as encrypting with a secret key

- Alice can send Bob an authenticated message by encrypting it with her secret key:

$$\{m\}_{SK(Alice)}$$

- Bob can decrypt this message with Alice's public key, and verify that Alice sent it

A Sample Protocol

Suppose agents a and b want to establish a cryptographic session key with the help of a trusted server s

We could arrange for s to generate a key K and have it distributed to both agents

1. $a \rightarrow s : a, s, b$
2. $s \rightarrow a : s, a, b, K$
3. $a \rightarrow b : a, b, K$

A Sample Protocol

Suppose agents a and b want to esta ion
key with the help of a trusted serve

We could arrange for s to generate a uted
to both agents

Every message has a (forgeable) from-field and to-field

1. a → s : a, s, b
2. s → a : s, a, b, K
3. a → b : a, b, K

A Sample Protocol

Suppose agents a and b want t n
key with the help of a trusted

We could arrange for s to gene ted
to both agents

What is wrong with this protocol?

1. a → s : a, s, b
2. s → a : s, a, b, K
3. a → b : a, b, K

A Second Attempt

Suppose that a and b share long-term keys shared(a,s) and shared (b,s) with s

Then the key delivery messages could be encrypted with those keys:

1. $a \rightarrow s : \; a, s, b$
2. $s \rightarrow a : \; s, a, b, \{K\}_{shared(a,s)}, \{K\}_{shared(b,s)}$
3. $a \rightarrow b : \; a, b, \{K\}_{shared(b,s)}$

This keeps the session key secret from eavesdroppers

The fact that the key delivery message is encrypted with shared (a,s) tells a that it was created by s

(1) Authentication via Shared Keys

If

(a) an agent a shares a key k with another agent s (and each knows that they share it)

(b) a receives a message encrypted with k, and

(c) a did not send the message herself

then a can deduce that s created the message, and that s intended the message for a

However...

This protocol is not secure against active opponents, who can intercept messages from the network, and introduce new messages

Consider the following attack:

$$
\begin{aligned}
o &\rightarrow s \;\; : o,\ s,\ b \\
s &\rightarrow o \;\; : s,\ o,\ b,\ \{K\}_{shared(s,o)},\ \{K\}_{shared(b,s)} \\
o_a &\rightarrow b : a,\ b,\ \{K\}_{shared(b,s)}
\end{aligned}
$$

Here, o is the opponent, and o_a represents o posing as a

b thinks he shares the key with a, but actually he shares the key only with o

This is a failure of both secrecy and authentication

Moreover...

There's another attack:

$$
\begin{aligned}
a &\rightarrow o_s : \quad a, s, b \\
o_a &\rightarrow s : \quad a, s, o \\
s &\rightarrow o_a : \quad s, a, o, \{K\}_{shared(a,s)}, \{K\}_{shared(o,s)} \\
o_s &\rightarrow a : \quad s, a, b, \{K\}_{shared(a,s)}, \{K\}_{shared(o,s)} \\
a &\rightarrow o_b : \quad a, b, \{K\}_{shared(o,s)}
\end{aligned}
$$

The opponent ends up knowing the key

This is a failure of both secrecy and authentication

Diagnosis and Correction

The problem is that the identity b in message 2, and the identity a in message 3 are essential to the meaning of the encrypted component, yet the opponent is able to separate them

This suggests that we should include those identities within the encryptions:

1. $a \rightarrow s : a, s, b$
2. $s \rightarrow a : s, a, \{b, K\}_{shared(a,s)}, \{a, K\}_{shared(b,s)}$
3. $a \rightarrow b : a, b, \{a, K\}_{shared(b,s)}$

Thus encryption can be used for binding objects together

Freshness

Suppose the opponent has overheard an old protocol session, and saves the key delivery messages:

$$\{b, K\}_{shared(a,s)} \quad \text{and} \quad \{a, K\}_{shared(b,s)}$$

Suppose he subsequently compromises session key K ; for example, because it is leaked, or because he breaks into agent a's computer. He can then replay the key delivery messages as follows:

$$\begin{array}{lll} a \rightarrow o_s & : & a, s, b \\ o_s \rightarrow a & : & s, a, \{b, K\}_{shared(a,s)}, \{a, K\}_{shared(b,s)} \\ a \rightarrow b & : & a, b, \{a, K\}_{shared(b,s)} \end{array}$$

Note that a and b think that K is a good key, but o knows it

Guaranteeing Recentness

When an agent a receives a message from another agent s, he will often want to be assured that s sent the message recently, rather than it being a replay of a message from an old execution.

One way to achieve this is through the use of nonces (large random numbers). If a sends s a nonce, and a subsequently receives the nonce back in a message, then a can deduce that the message (or, at least, the part of the message containing the nonce) was created recently.

Guaranteeing Recentness

a and b can be assured of the freshness of the session key by each creating a nonce, and by s including that nonce in the key delivery messages:

1. $a \rightarrow b : \; a, b, n_a$
2. $b \rightarrow s : \; b, s, a, n_a, n_b$
3. $s \rightarrow a : \; s, a, \{b, K, n_a\}_{shared(a,s)}, \{a, K, n_b\}_{shared(b,s)}$
4. $a \rightarrow b : \; a, b, \{a, K, n_b\}_{shared(b,s)}$

What About Authentication?

Neither a nor b receives any guarantee that the other agent is involved in the protocol - the protocol does not provide authentication

The intruder can imitate the responder b as follows:

$$
\begin{aligned}
a &\rightarrow o_b &&: a, b, n_a \\
o_b &\rightarrow s &&: b, s, a, n_a, n_o \\
s &\rightarrow a &&: s, a, \{b, K, n_a\}_{shared(a,s)}, \{a, K, n_o\}_{shared(b,s)} \\
a &\rightarrow o_b &&: a, b, \{a, K, n_b\}_{shared(b,s)}
\end{aligned}
$$

What About Authentication?

The intruder can also imitate the initiator a as follows:

$$
\begin{aligned}
o_a \rightarrow b &: \; a, b, n_o \\
b \rightarrow s &: \; b, s, a, n_o, n_b \\
s \rightarrow o_a &: \; s, a, \{b, K, n_o\}_{shared(a,s)}, \{a, K, n_b\}_{shared(b,s)} \\
o_a \rightarrow b &: \; a, b, \{a, K, n_b\}_{shared(b,s)}
\end{aligned}
$$

Achieving Authentication

We can achieve authentication, and each agent can prove they know the key, by having a nonce exchange:

1. $a \rightarrow b : \; a, b, n_a$
2. $b \rightarrow s : \; b, s, a, n_a, n_b, n_b'$
3. $s \rightarrow a : \; s, a, n_b', \{b, K, n_a\}_{shared(a,s)}, \{a, K, n_b\}_{shared(b,s)}$
4. $a \rightarrow b : \; a, b, n_a', \{a, K, n_b\}_{shared(b,s)}, \{a, n_b'\}_K$
5. $b \rightarrow a : \; b, a, \{n_a', b\}_K$

The Yahalom Protocol

1. $a \rightarrow b : \; a, b, n_a$
2. $b \rightarrow s : \; b, s, \{a, n_a, n_b\}_{shared(b,s)}$
3. $s \rightarrow a : \; s, a, \{b, K, n_a, n_b\}_{shared(a,s)}, \{a, K\}_{shared(b,s)}$
4. $a \rightarrow b : \; a, b, \{a, K\}_{shared(b,s)}, \{n_b\}_K$

- The Yahalom Protocol establishes K as a shared secret
- The Yahalom Protocol authenticates a to b, and vice versa

Authentication in Yahalom

Because of the use of the shared keys:

- s can deduce that b created the encrypted component of message 2
- a can deduce that s created the first encrypted component of message 3
- b can deduce that s created the first encrypted component of message 4

Recentness in Yahalom

Because of the use of nonces:

- a can deduce that s sent the first encrypted component of message 3 recently
- b can deduce that the second part of message 4 was created recently

Trust in Yahalom

Because s is assumed to be trustworthy, and in particular creates good cryptographic keys K:

- a can deduce that the key he receives in message 3 is a good key to share with b
- a can deduce that b has recently been running the protocol with a
- b can deduce that the key he receives in message 4 is a good key to share with a
- b can hence deduce that a sent the second encrypted component of message 4, and that this component was created recently

Key Confirmation in Yahalom

The Yahalom Protocol assures b that a has received key K

The Yahalom protocol does not assure a that b has received K

(2) Authentication via Public Keys

If an agent a sees a message encrypted with b's secret key, then she can deduce that b created the message

If a sends a message encrypted with b's public key, and which contains a secret value v, and subsequently receives v back, then a can deduce that b decrypted the message

Needham-Schroeder Public Key Protocol

1. $a \rightarrow b : \; a, b, \{a, n_a\}_{PK(b)}$
2. $b \rightarrow a : \; b, a, \{n_a, n_b\}_{PK(a)}$
3. $a \rightarrow b : \; a, b, \{n_b\}_{PK(b)}$

The protocol aims to authenticate each agent to the other, and to establish a pair of shared secrets n_a and n_b

An Attack Against Needham-Schroeder

This protocol is subject to the following attack:

$$
\begin{array}{lcl}
a \rightarrow o & : & a,\ o,\ \{a,\ n_a\}_{PK(o)} \\
o_a \rightarrow b & : & a,\ b,\ \{a,\ n_a\}_{PK(b)} \\
b \rightarrow o_a & : & b,\ a,\ \{n_a,\ n_b\}_{PK(a)} \\
o \rightarrow a & : & o,\ a,\ \{n_a,\ n_b\}_{PK(a)} \\
a \rightarrow o & : & a,\ o,\ \{n_b\}_{PK(o)} \\
o_a \rightarrow b & : & o,\ b,\ \{n_b\}_{PK(b)}
\end{array}
$$

A Variant of Needham-Schroeder

We can prevent the attack by modifying Needham-Schroeder:

1. $a \rightarrow b : \; a, b, \{a, n_a\}_{PK(b)}$
2. $b \rightarrow a : \; b, a, \{b, n_a, n_b\}_{PK(a)}$
3. $a \rightarrow b : \; a, b, \{n_b\}_{PK(b)}$

The Attack under the Variant

Trying the attack, we see where it would fail:

$$
\begin{array}{lcl}
a \rightarrow o & : & a,\ o,\ \{a,\ n_a\}_{PK(o)} \\
o_a \rightarrow b & : & a,\ b,\ \{a,\ n_a\}_{PK(b)} \\
b \rightarrow o_a & : & b,\ a,\ \{b,\ n_a,\ n_b\}_{PK(a)} \\
o \rightarrow a & : & o,\ a,\ \{b,\ n_a,\ n_b\}_{PK(a)}
\end{array}
$$

Here a would reject the message, since the from-field of the message says o, but the content of the encrypted message says that the message came from b

- a should check that, and abort

Needham-Schroeder Shared Key Protocol

1. $a \rightarrow s : \quad a, s, b, n_a$
2. $s \rightarrow a : \quad s, a, \{n_a, b, K, \{K, a\}_{shared(b,s)}\}_{shared(a,s)}$
3. $a \rightarrow b : \quad a, b, \{K, a\}_{shared(b,s)}$
4. $b \rightarrow a : \quad b, a, \{n_b\}_K$
5. $a \rightarrow b : \quad a, b, \{n_b - 1\}_K$

An Attack

Suppose the opponent has observed a previous execution of the protocol between a and b, and stored message 3:

$$\{K,\ a\}_{shared(b,s)}$$

Suppose further the opponent compromises key K, maybe by breaking into b's computer.

The intruder could then attack b as follows:

$$o_a \rightarrow b : a,\ b,\ \{K,\ a\}_{shared(b,s)}$$
$$b \rightarrow o_a : b,\ a,\ \{n_b\}_K$$
$$o_a \rightarrow b : a,\ b\ \{n_b - 1\}_K$$

Key Compromises

We have to accept that key compromise might not be avoidable

- Ensure that if an old key is compromised, then the opponent cannot replay it to cause a failure of authentication

Protocols should be designed so that each agent who receives a key also receives some evidence that the key is fresh - it has not been used in a previous instance of the protocol.

- This evidence could be via appropriate use of a nonce that the agent knows is fresh, or via a timestamp

The Kerberos Protocol

Kerberos was invented by MIT as part of Project Athena.

It aims to authenticate (users of) clients and servers to one another, and to establish session keys between them, to allow secure transfer of data.

The Kerberos protocol defines four types of agent:

(a) Clients

(b) Servers

(c) Ticket granting servers (TGSs), that give tickets to clients; these tickets can be used to authenticate the client to a server, and to establish a session key

(d) Kerberos, which gives ticket granting tickets (TGTs) to clients, with which they can authenticate themselves to the TGS

Overview of Kerberos

The Kerberos protocol has three parts:

(1) A client obtains a TGT from Kerberos

(2) The client uses the TGT to obtain a ticket for a particular server from the TGS

(3) The client uses this ticket to authenticate itself to the server and establish a session key

Kerberos Keys

The Kerberos protocol uses two types of keys:

- Each agent a has a long term key key(a)
  - Kerberos knows the long term keys of clients and TGSs
  - Each TGS knows the long term key of local servers
- Clients can share session keys with either servers or TGSs
  - We write $K_{c,s}$ for a session key intended to be shared between client c and server or TGS s

All keys are symmetric (normally DES)

Kerberos Tickets

A ticket that client c can use to authenticate itself to s has form:

$$T_{c,s} = \{s, c, t, K_{c,s}\}_{key(s)}$$

Here, t is a timestamp, set to the time at which the ticket is created, and used to verify that a ticket is still valid.

If s is a TGS then this ticket is produced by Kerberos; if s is a normal server then this ticket is produced by a TGS.

- Note that only s can decrypt $T_{c,s}$

Getting a Ticket-Granting Ticket

A client obtains a TGT by sending its identity and the identity of an appropriate TGS to Kerberos.

Kerberos returns a session key and TGT:

1. $c \rightarrow kerb$: $c, kerb, c, tgs$
2. $kerb \rightarrow c$: $kerb, c, \{T_{c,tgs}, K_{c,tgs}\}_{key(c)}$

key(c) is formed as a one-way hash of c's password

The user needs to supply the correct password in order for the client to obtain the session key $K_{c,tgs}$

Getting a Ticket

A client can request a ticket for a particular server from a TGS:

3. $c \rightarrow tgs : \; c, tgs, T_{c,tgs}, \{c, t\}_{Kc,tgs}$
4. $tgs \rightarrow c : \; tgs, c, \{T_{c,s}, K_{c,s}\}_{Kc,tgs}$

The TGS extracts the key $K_{c,tgs}$ from $T_{c,tgs}$

This step can repeated multiple times (with the same TGT) to obtain tickets for different servers

Requesting a Service

Finally, clients can request a service from a server by sending the ticket:

$$5.\ \ c \rightarrow s:\ \ c,\ s,\ T_{c,s},\ \{c,\ t\}_{Kc,s}$$

$K_{c,s}$ can then be used to transfer information

This step can be repeated multiple times, with the same ticket

Analyzing Security Protocols

I have given you a sample of security protocols achieving secrecy and authentication goals

- Many more exist and have been studied
- "A Survey of Authentication Protocol Literature", by J. Clarke and J. Jacob

Question: how do you prove that a protocol satisfies its secrecy or authentication goals? Many automated techniques developed:

- Language-based: spi-calculus, Cryptyc, ...
- Model-checking-based: AVISS, Murφ, ...
- Logic-based: BAN logic, Paulson's inductive assertions, ...